

ĐẠI HỌC QUỐC GIA TP HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



BÁO CÁO THỰC HÀNH



LAB: 3

TIẾN TRÌNH VÀ TIỂU TRÌNH

Lớp thực hành môn: HỆ ĐIỀU HÀNH
– **IT007.M11.CLC.1**



Sinh viên thực hiện:

MSSV: 20520322

Họ và tên: Nguyễn Thị Mỹ Trân



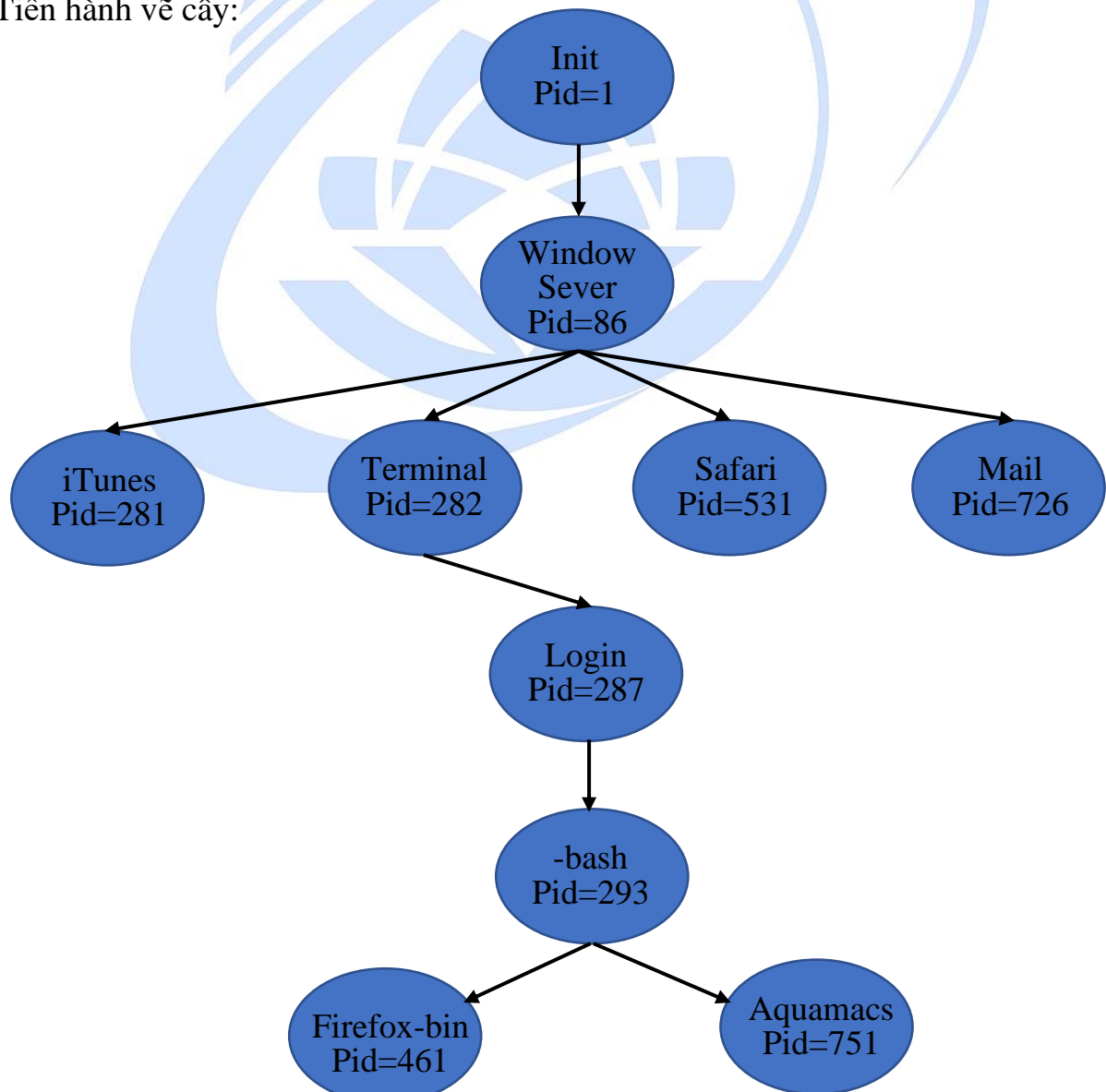
3.5 Bài tập ôn tập

1. Môi quan hệ cha-con giữa các tiến trình

a. Vẽ cây quan hệ parent-child của các tiến trình bên dưới:

UID	PID	PPID	COMMAND
88	86	1	WindowServer
501	281	86	iTunes
501	282	86	Terminal
0	287	282	login
501	461	293	firefox-bin
501	531	86	Safari
501	726	86	Mail
501	751	293	Aquamacs
501	293	287	-bash

Tiến hành vẽ cây:





b. Trình bày cách sử dụng lệnh ps để tìm tiến trình cha của một tiến trình dựa vào PID của nó.

- - Lệnh ps dùng để liệt kê **chi tiết** cả các tiến trình

```
mytran@mytran-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2644 pts/0        00:00:00 bash
 2650 pts/0        00:00:00 ps
```

- Lệnh ps -f là lệnh liệt kê đầy đủ và chi tiết hơn lệnh ps.

```
mytran@mytran-VirtualBox:~$ ps -f
  UID          PID    PPID  C   STIME TTY          TIME CMD
mytran        2644    2636  0   23:09 pts/0        00:00:00 bash
mytran        2665    2644  0   23:10 pts/0        00:00:00 ps -f
mytran@mytran-VirtualBox:~$
```

- Các thông số được trình bày dưới đây:

Cột	Mô tả
UID	ID người sử dụng, mà tiến trình buộc phải sở hữu.
PID	ID của tiến trình.
PPID	ID của tiến trình cha.
C	CPU sử dụng tiến trình.
STIME	Thời gian bắt đầu tiến trình.
TTY	Kiểu terminal liên kết với tiến trình.
CMD	Lệnh bắt đầu tiến trình này.

- Có những tiến trình khác mà có thể chạy song song với lệnh ps



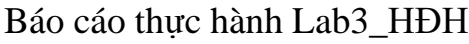
Tùy chọn	Mô tả
-a	Chỉ thông tin về tất cả người dùng.
-x	Chỉ thông tin các tiến trình mà không có termial.
-u	Chỉ thông tin thêm vào chức năng -f
-e	Hiển thị thông tin mở rộng

- Ta có thể tìm tiến trình cha bằng cách nhìn vào cột PID rồi sau đó nhìn qua cột PPID là cột chứa ID của tiến trình của tiến trình ID tiến trình cột PID

c. Tìm hiểu và cài đặt lệnh pstree (nếu chưa được cài đặt), sau đó trình bày cách sử dụng lệnh này để tìm tiến trình cha của một tiến trình dựa vào PID của nó.

- Lệnh pstree liệt kê tiến trình theo dạng cây thay vì theo dạng danh sách như lệnh ps

```
mytran@mytran-VirtualBox:~$ pstree
systemd--ModemManager--2*[{ModemManager}]
systemd--NetworkManager--2*[{NetworkManager}]
systemd--accounts-daemon--2*[{accounts-daemon}]
systemd--acpid
systemd--avahi-daemon--avahi-daemon
systemd--colord--2*[{colord}]
systemd--cron
systemd--cups-browsed--2*[{cups-browsed}]
systemd--cupsd--dbus
systemd--dbus-daemon
systemd--fwupd--4*[{fwupd}]
systemd--gdm3--gdm-session-wor--gdm-x-session--Xorg--5*[{Xorg}]
systemd--gdm3--gdm-session-wor--gdm-x-session--gnome-session-b--ssh-agent
systemd--gdm3--gdm-session-wor--gdm-x-session--2*[{gnome-+
systemd--gdm3--gdm-session-wor--2*[{gdm-x-session}]
systemd--gdm3--2*[{gdm3}]
systemd--gnome-keyring-d--3*[{gnome-keyring-d}]
systemd--2*[kerneloops]
systemd--networkd-dispat
systemd--polkitd--2*[{polkitd}]
systemd--rsyslogd--2*[{rsyslogd}]
```



- ```
nytran@mytran-VirtualBox:~$ pstree -p
systemd(1)─ModemManager(640)├──{ModemManager}(653)
│ └──{ModemManager}(657)
│ ─NetworkManager(550)├──{NetworkManager}(595)
│ └──{NetworkManager}(620)
│ ─accounts-daemon(538)├──{accounts-daemon}(543)
│ └──{accounts-daemon}(616)
│ ─acpid(539)
│ ─avahi-daemon(542)─avahi-daemon(594)
│ ─colord(1111)├──{colord}(1112)
│ │ └──{colord}(1114)
│ ─cron(545)
│ ─cups-browsed(621)├──{cups-browsed}(642)
│ └──{cups-browsed}(643)
│ ─cupsd(546)─dbus(613)
│ ─dbus-daemon(549)
│ ─fwupd(1856)├──{fwupd}(1857)
│ │ ├──{fwupd}(1858)
│ │ ├──{fwupd}(1859)
│ │ └──{fwupd}(1860)
│ ─gdm3(652)├──gdm-session-wor(1143)├──gdm-x-session(1215)├──Xorg(121+
│ │ ├──gnome-se+
│ │ ├──gdm-x-s+
│ │ └──gdm-x-s+
│ │ ├──{gdm-session-wor}(1144)
│ │ └──{gdm-session-wor}(1145)
│ ├──{gdm3}(660)
│ └──{gdm3}(661)
│ ─gnome-keyring-d(1162)├──{gnome-keyring-d}(1163)
│ │ ├──{gnome-keyring-d}(1164)
│ │ └──{gnome-keyring-d}(1370)
```

- ## 2. Chạy chương trình và đưa ra kết quả rồi giải thích



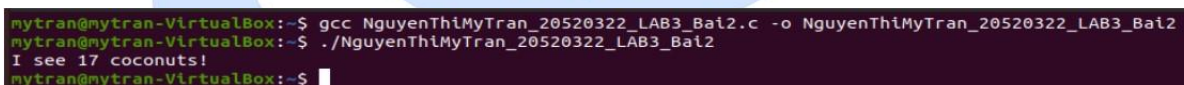
## Tạo 1 file.c có tên: NguyenThiMyTran\_20520322\_LAB3\_Bai2.c

### • Nội dung file:

```
/*#####
University of Information Technology #
IT007 Operating System #
<Your name>, <your Student ID> #
File: exercise_2.c #
#####*/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main(){
 pid_t pid;
 int num_coconuts = 17;
 pid = fork();
 if(pid == 0) {
 num_coconuts = 42;
 exit(0);
 } else {
 wait(NULL); /*wait until the child terminates */
 }
 printf("I see %d coconuts!\n", num_coconuts);
 exit(0);
}
```

### • Kết quả chạy chương trình



```
mytran@mytran-VirtualBox:~$ gcc NguyenThiMyTran_20520322_LAB3_Bai2.c -o NguyenThiMyTran_20520322_LAB3_Bai2
mytran@mytran-VirtualBox:~$./NguyenThiMyTran_20520322_LAB3_Bai2
I see 17 coconuts!
mytran@mytran-VirtualBox:~$
```

### • Giải thích

1. Đặt biến pid với kiểu dữ liệu pid\_t
2. Gán biến num\_coconuts = 17 kiểu số nguyên.
3. Tạo tiến trình pid = fork();
4. In ra dòng “I see 17 coconuts” vì pid lúc này > 0 sẽ vào else  
{wait (NULL);} vậy giá trị coconuts là 17.





**3. Trong phần thực hành, các ví dụ chỉ sử dụng thuộc tính mặc định của pthread, hãy tìm hiểu POSIX thread và trình bày tất cả các hàm được sử dụng để làm thay đổi thuộc tính của pthread, sau đó viết các chương trình minh họa tác động của các thuộc tính này và chú thích đầy đủ cách sử dụng hàm này trong chương trình. (Gợi ý các hàm liên quan đến thuộc tính của pthread đều bắt đầu bởi: pthread\_attr\_\*)**

#### **POSIX thread**

- POSIX thread (pthread) sẽ được sử dụng để lập tiểu trình. Nó cho phép chúng ta tạo ra các ứng dụng chạy song song theo luồng, phù hợp với các hệ thống đa bộ xử lý. POSIX (Portable Operating Systems Interface) là mô tả các API (Application Programming Interface) bao gồm hàm và chức năng của chúng.
- Các hàm liên quan đến thuộc tính của thread đều bắt đầu bởi:  
pthread\_attr\_\*
- Thư viện pthread.h có các hàm sau:



| Attribute           | Value                   | Result                                                                                                                                           |
|---------------------|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>scope</i>        | PTHREAD_SCOPE_PROCESS   | New thread is unbound - not permanently attached to LWP.                                                                                         |
| <i>detachstate</i>  | PTHREAD_CREATE_JOINABLE | Exit status and thread are preserved after the thread terminates.                                                                                |
| <i>stackaddr</i>    | NULL                    | New thread has system-allocated stack address.                                                                                                   |
| <i>stacksize</i>    | 1 megabyte              | New thread has system-defined stack size.                                                                                                        |
| <i>priority</i>     |                         | New thread inherits parent thread priority.                                                                                                      |
| <i>inheritsched</i> | PTHREAD_INHERIT_SCHED   | New thread inherits parent thread scheduling priority.                                                                                           |
| <i>schedpolicy</i>  | SCHED_OTHER             | New thread uses Solaris-defined fixed priority scheduling; threads run until preempted by a higher-priority thread or until they block or yield. |

- Minh họa scope trong thư viện pthread

```
#include <pthread.h>
```

```
int pthread_attr_setscope(pthread_attr_t *attr, int scope);
```

```
int pthread_attr_getscope(const pthread_attr_t *attr, int *scope);
```



**Note -**

Both thread types are accessible only within a given process.

**Prototype :**

```
int pthread_attr_setscope(pthread_attr_t *tattr, int scope);
#include <pthread.h>

pthread_attr_t tattr;
int ret;

/* bound thread */
ret = pthread_attr_setscope(&tattr, PTHREAD_SCOPE_SYSTEM);

/* unbound thread */
ret = pthread_attr_setscope(&tattr, PTHREAD_SCOPE_PROCESS);
```

Notice that there are three function calls in this example: one to initialize the attributes, one to set any variations from the default attributes, and one to create the pthreads.

```
#include <pthread.h>

pthread_attr_t attr;
pthread_t tid;
void start_routine;
void arg;
int ret;

/* initialized with default attributes */
ret = pthread_attr_init (&tattr);

/* BOUND behavior */
ret = pthread_attr_setscope(&tattr, PTHREAD_SCOPE_SYSTEM);
ret = pthread_create (&tid, &tattr, start_routine, arg);
```

**Return Values**

pthread\_attr\_setscope() returns zero after completing successfully. Any other returned value indicates that an error occurred. If the following conditions occur, the function fails and returns the corresponding value.

- Minh họa thuộc tính priority
  - Cách đặt độ ưu tiên của một luồng

```
res=pthread_attr_setschedpolicy(&thread_attr,
 SCHED_OTHER);

if(res!=0) {
 perror("Set thread policy error");
 exit(EXIT_FAILURE);
}

/*Lấy về độ ưu tiên Tối đa và Tối thiểu của luồng*/
max_priority=sched_get_priority_max(SCHED_OTHER);
min_priority=sched_get_priority_min(SCHED_OTHER);
```

- Minh họa hàm setstack và hàm getstack trong ví dụ sau:
  - Nội dung chương trình minh họa:

```
#include <stdio.h>
```



```
#include <stdlib.h>

#include <pthread.h>

int main()

{ // Khai báo biến kích thước cho stack

 size_t stksize;

 // Khai báo biến cho stack

 pthread_attr_t atr;

 // sử dụng pthread để lấy kích thước và set kích thước

 // Giá trị của stack size ban đầu

 pthread_attr_getstacksize(&atr, &stksize);

 // In ra kích thước stack cũ

 printf("Kích thước stack cũ - > %d\n", stksize);

 // Đặt lại stack size mới

 pthread_attr_setstacksize(&atr, 275222233);

 pthread_attr_getstacksize(&atr, &stksize);

 // In ra stack size mới

 printf("Kích thước stack mới-> %d\n", stksize);

 return 0;
```



}

- **Kết quả chạy chương trình:**

Current stack size - > 4196464

New stack size-> 275222233

- **Giải thích:**

Hàm **pthread\_attr\_getstacksize** để lấy kích thước stack hiện tại.

Hàm **pthread\_attr\_setstacksize** để set kích thước stack mới.

**4. Viết chương trình làm các công việc sau theo thứ tự:**

**a. In ra dòng chữ: “Welcome to IT007, I am!”**

**b. Mở tệp abcd.txt bằng vim editor**

**c. Tắt vim editor khi người dùng nhấn CTRL+C**

**d. Khi người dùng nhấn CTRL+C thì in ra dòng chữ: “You are pressed CTRL+C! Goodbye!”**

**Viết chương trình lưu tại file: NguyenThiMyTran\_20520322\_LAB3\_Bai4.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
```

```
int loop=1;
```

```
void on_sigint()
```

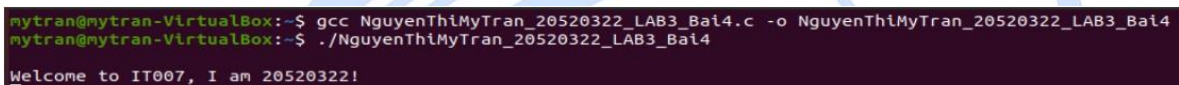


```
{
 printf("You are press CTRL+C! Goodbye");
 loop=0;
}

int main()
{
 loop=1;
 printf("\nWelcome to IT007, I am 20520322!\n");
 system("gedit abcd.txt");
 signal(SIGINT,on_sigint);
 while(loop){}
 return 0;
}
```

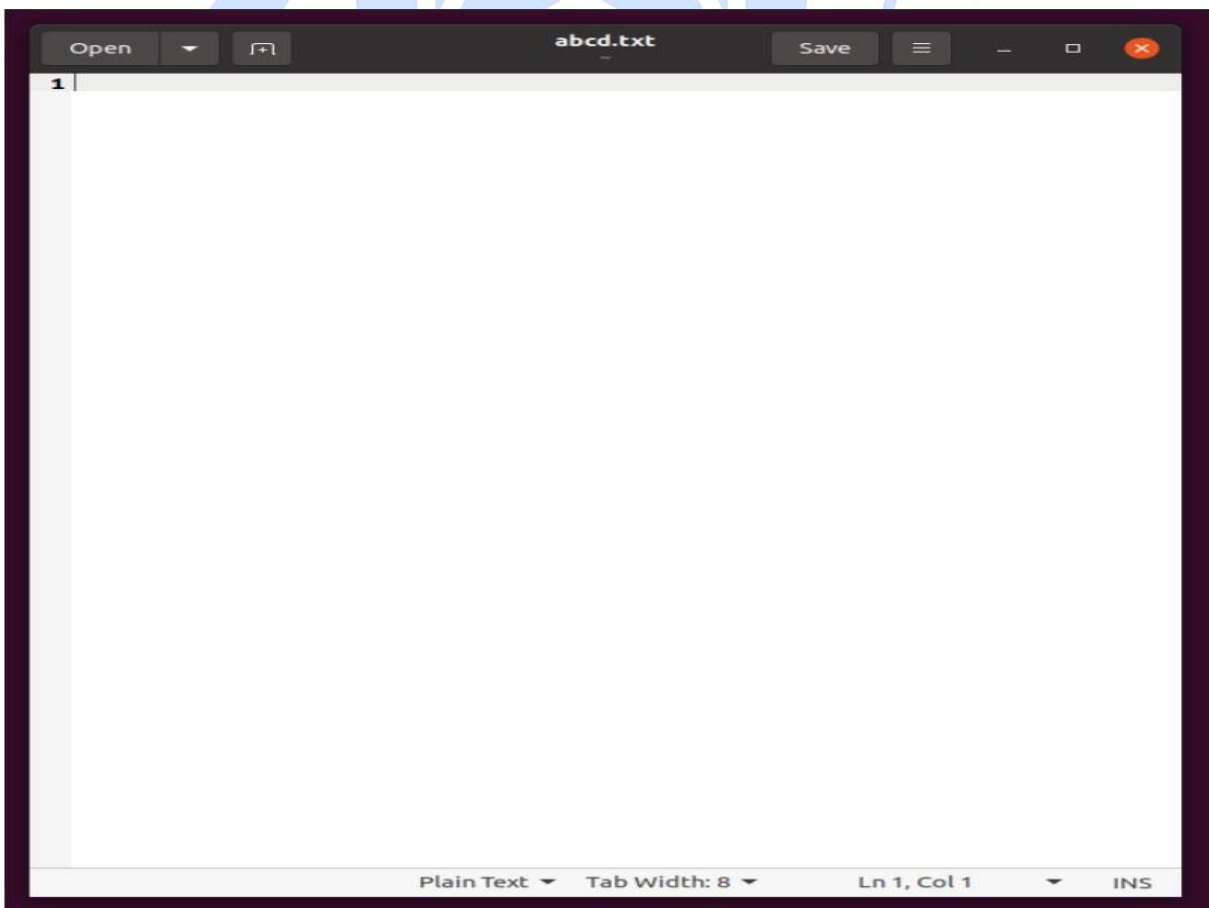
### Kết quả chạy chương trình:

- a. In ra dòng chữ “Welcome to IT007, I am 20520322!” bằng lệnh printf (“Welcome to IT007, I am 20520322!”);



```
nytran@nytran-VirtualBox:~$ gcc NguyenThiMyTran_20520322_LAB3_Bai4.c -o NguyenThiMyTran_20520322_LAB3_Bai4
nytran@nytran-VirtualBox:~$./NguyenThiMyTran_20520322_LAB3_Bai4
Welcome to IT007, I am 20520322!
```

- b. Tiến hành mở file abcd.txt bằng lệnh system(“gedit abcd.c”);





**c. Tắt vim khi người dùng nhấn CTRL + C**

```
mytran@mytran-VirtualBox:~$ gcc NguyenThiMyTran_20520322_LAB3_Bai4.c -o NguyenThiMyTran_20520322_LAB3_Bai4
mytran@mytran-VirtualBox:~$./NguyenThiMyTran_20520322_LAB3_Bai4
Welcome to IT007, I am 20520322!
^C
```

**d. Khi người dùng nhấn CTRL+C thì in ra dòng chữ: “You are pressed CTRL+C! Goodbye!”**

```
mytran@mytran-VirtualBox:~$ gcc NguyenThiMyTran_20520322_LAB3_Bai4.c -o NguyenThiMyTran_20520322_LAB3_Bai4
mytran@mytran-VirtualBox:~$./NguyenThiMyTran_20520322_LAB3_Bai4
Welcome to IT007, I am 20520322!
^C^CYou are press CTRL+C! Goodbye!mytran@mytran-VirtualBox:~$
```

HẾT